Roughly speaking, "best practices" can be divided into two categories - mechanics and design. Good Python mechanics adhere to [PEP 8](https://www.python.org/dev/peps/pep-0008/) - the style guide for Python code - and [PEP 257](https://www.python.org/dev/peps/pep-0257/) - docstring conventions. These address rules for naming, spacing, commenting, and several common programming patterns. Additionally, good general programming mechanics that you are expected to follow include removing starter code markings, removing extraneous print statements, and documenting your code. Python design refers to the higher-level organization of your code - the interfaces and implementation boundaries defined by your code objects.

* Review core Pythonic design, including *object-oriented systems thinking*
* Write consistent, scalable code using the [PEP8](https://www.python.org/dev/peps/pep-0008/) style guide and the [Pycodestyle](https://pypi.org/project/pycodestyle/) linter
* Create well-documented code using *Docstrings*, and *Doctests*

|  |  |
| --- | --- |
| **Criteria** | **Meets Specifications** |
| Write code that is PEP compliant and follows common programming best practices. | The code adheres to the [PEP 8 style guide](https://www.python.org/dev/peps/pep-0008/) and follows common best practices, including:   * [Variable and function names are clear](https://www.python.org/dev/peps/pep-0008/#prescriptive-naming-conventions). * The code is DRY (Don’t Repeat Yourself) and methods demonstrate the principle of composition. |
| Create basic documentation. | All included docstrings and comments adhere to [PEP 8 standards](https://www.python.org/dev/peps/pep-0008/#comments)  A README file is included in the project root and includes:   * an overview of the project. * instructions for setting up and running the program. * a brief description of the roles-and-responsibilities of all sub-modules including dependencies and examples of how to use the module. |
| Consume public libraries using virtual environments. | The code makes use of public libraries using a virtual environment.  All required dependencies are listed in the root requirements.txt file which was created using the $ pip freeze > requirements.txt bash command.  The program runs with no errors.  (Optional) If git is used, the virtual environment directory is added to the .gitignore file. |
| Implement basic Python exception handling. | If the program encounters a common error case (e.g. attempting to load an incompatible filetype), it throws an exception.  All exceptions include a human-readable message.  (Optional) Make your exception handling even more awesome:   * Define custom exception classes for different types of exceptions—for things like \*Invalid File, Invalid Text Input (e.g. too long) * Use os.walk to automatically discover ingestible files in a directory |
| Create Python modules. | Classes are organized into multiple directories, with related classes being placed together.  Each module directory includes a proper \_\_init\_\_.py file. |

Style (Mechanics)

|  |  |
| --- | --- |
| **Criteria** | **Meets Specifications** |
| Produce Python code that satisfies PEP 8. | Submitted code follows the guidelines of PEP 8 - the Style Guide for Python. |
| Write Python comments that satisfy PEP 257. | Submitted code follows the docstring conventions of PEP 257.  Each module contains a module-level comment describing the purpose of the module. Complex functions, classes, and methods include a docstring annotating the primary action of the callable in the imperative mood, any additional clarifications, followed by descriptions of parameters and return values. |
| Submit code free from starter code markings. | There are no # TODO comments left in the submitted code. Portions of comments that say # ELABORATE have been filled in with a description of the corresponding code. |

Style (Design)

|  |  |
| --- | --- |
| **Criteria** | **Meets Specifications** |
| Divide attributes appropriately among the NearEarthObject class and the CloseApproach class. | Attributes of NearEarthObjects and CloseApproaches are captured in the constructor from the supplied arguments.  Instances of NearEarthObject don’t have attributes of individual close approaches.  Instances of CloseApproachdon’t have attributes of the associated NEO (except for the primary designation needed to initially link the close approach to its NEO. |
| Attach functionality to classes only when appropriate. | Standalone functions are used when the functional operation doesn’t depend on external state and does not conceptually belong on an object. |
| Design a coherent system to filter objects by user-specified criteria. | The logic backing the filter system is consistent and doesn’t contain excess duplicated code. |
| Consume and produce streams of data as iterables. | Represents concrete data (buffered file contents, static collections of NEOs or close approaches, auxiliary data structures) as concrete.  Represents streaming data (close approaches that match criteria, limited stream of results) as streaming. |